UNIVERSITÀ DEGLI STUDI
DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA DELLE TELECOMUNICAZIONI

CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E TECNOLOGIEDELL'INFORMAZIONE

Tesi di Laurea Triennale

Giugno 2021

"PROGETTAZIONE E SVILUPPO DI UNA SISTEMA EMBEDDED PER IL CONTROLLO REMOTO DEGLI ACCESSI"

Candidato: Fabio Mora

Relatore: Prof. Riccardo Berta

Correlatore: Dott. Luca Lazzaroni

Sommario

Lo scopo di questa tesi è quello di sviluppare un sistema embedded per la rilevazione di ingressi e uscite da un luogo pubblico come, ad esempio, nel caso scelto, un supermercato. Per acquisire i dati sono stati utilizzati due sensori PIR (Passive InfraRed). Per l'elaborazione delle misurazioni è stata usata una scheda di sviluppo con microcontrollore ESP32 e modulo WiFi integrato. Alla board è stato collegato un display LCD per la visualizzazione in loco delle persone presenti all'interno dell'esercizio commerciale. I dati raccolti vengono poi inviati al cloud sfruttando le API fornite dal framework Measurify. Successivamente è stata sviluppata un'applicazione utilizzando un framework cross-platform chiamato Flutter che utilizza il linguaggio Dart. Come ambiente di sviluppo è stato utilizzato Visual Studio Code istallando le estensioni per il supporto del linguaggio Dart e per il framework Flutter. Attraverso l'applicazione è possibile controllare da remoto il numero di persone presenti in un luogo pubblico sia nell'istante in un cui si sta eseguendo la ricerca, sia in un momento a nostro piacimento inserendo data e ora con i comandi predisposti.

1. Introduzione

L'idea alla base di questa tesi è nata a maggio 2020 durante il lockdown nazionale dovuto alla pandemia mondiale da SARS-CoV-2 (Severe Acute Distress Respiratory Syndrome Coronavirus 2) iniziato il 9 marzo 2020 e conclusosi il 18 maggio 2020. Durante questo periodo uscire di casa era possibile solamente per motivi di salute o per bisogni di prima necessità, quali fare la spesa, andare in farmacia etc. Al fine di ridurre al minimo gli assembramenti e quindi la possibilità di contagio, cercare di recarsi in luoghi pubblici in orari meno affollati era diventata una priorità. Da qui l'idea di creare un'applicazione che permettesse, attraverso lo smartphone, di monitorare l'afflusso di persone nei luoghi pubblici dove acquistare beni di prima necessità, e quindi capire il momento più opportuno e "sicuro" per uscire di casa, evitando così code e assembramenti. Questa applicazione permette in primo luogo di visualizzare il numero di persone presenti nel punto vendita in quell'istante e in secondo luogo di consultare l'andamento degli accessi nei giorni precedenti agli orari specificati dall'utente.

Il progetto sviluppato nell'ambito di questo lavoro si inserisce in un contesto di forte attualità rappresentato dall'acronimo **IOT (Internet Of Things).**

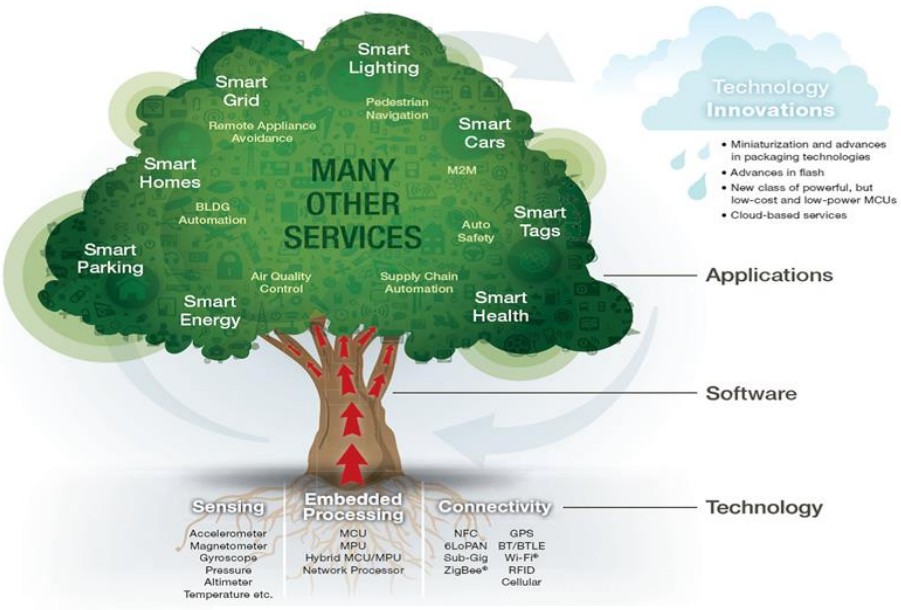

Figura 1: Rappresentazione grafica degli strati di IoT

Per "Internet of Things" o "Internet delle Cose" si intende quel percorso nello sviluppo tecnologico in base al quale, attraverso la rete Internet, potenzialmente ogni oggetto dell'esperienza quotidiana acquista una sua identità nel mondo digitale.

L'IoT si basa sull'idea di oggetti "intelligenti" tra loro interconnessi in modo da scambiare le informazioni possedute, raccolte e/o elaborate. L'IoT è abilitato da tre "strati" che si possono definire come:
- strato tecnologico
- strato del software
-strato delle applicazioni
Il primo, quello alla base, si occupa:
-della raccolta dei dati attraverso i sensori
-della prima elaborazione di quest'ultimi attraverso l'utilizzo di piattaforme come Arduino, Rapsberry o nel nostro caso ESP32
-della trasmissione dei dati ottenuti attraverso tecnologie di connessione wireless quali per esempio WiFi e Bluetooth.

Il secondo livello, quello software, si occupa dell'elaborazione e standardizzazione dei dati che riceve dallo strato tecnologico. Questi dati poi vengono messi a disposizione dell'ultimo strato che è quello delle applicazioni e dei servizi.

Gli ambiti applicativi di questo paradigma sono potenzialmente illimitati. Esempi di settori che traggono un notevole vantaggio dalle tecnologie in ambito IOT sono quelli dell'auto e dei trasporti, la domotica, l'industria e l'edilizia con gli Smart Building.

Questo progetto di tesi si è articolato in due parti principali:

1. Il sistema embedded per la raccolta dati

2. L'applicazione smartphone

Per quanto riguarda la prima parte si è utilizzato l'IDE di Arduino configurato in modo da poter essere utilizzato anche con altre schede di sviluppo diverse da Arduino stesso. Gli step successivi sono stati la progettazione e realizzazione dello **sketch**, che è l'elenco di istruzioni che la scheda dovrà eseguire, per la gestione della board con microcontrollore e dei sensori. Ad esempio, sono state inserite le istruzioni per gestire i sensori PIR, per abilitare la connessione ad Internet tramite WiFi e quella con le **API di Measurify** che permettono di salvare i dati nel cloud. Quindi grazie a questo sistema integrato i dati vengono raccolti e inviati al cloud da dove poi saranno prelevati, sempre attraverso la chiamata alle API di Measurify, per essere utilizzati nell'applicazione smartphone.

Per la seconda parte sono stati utilizzati l'ambiente di sviluppo **Visual Studio Code** e l'emulatore fornito da Android Studio. L'applicazione è stata divisa in tre schermate. Nella prima vengono chiesti lo username e la password che permetteranno di accedere all'applicazione e saranno usati da quest'ultima per autenticarsi e accedere alle API di Measurify. Se ci dovessero essere degli errori nella digitazione di username e password o uno dei campi fosse lasciato vuoto, opportuni messaggi di errore saranno visualizzati. Superata la schermata di login si accede alla seconda pagina dove è visibile un elenco di luoghi dei quali è possibile monitorare l'affollamento. Una volta scelto il locale di interesse, il software reindirizza l'utente alla terza pagina. In quest'ultima si può visualizzare il numero di *persone presenti in quell' istante* con un semplice click sull'apposito bottone. La data e l'ora sono precompilati con i valori del momento della pressione del bottone. Se invece lo scopo è conoscere l'*affluenza* in un momento specifico dei giorni precedenti è possibile scegliere la data e l'ora utilizzando le icone predisposte. L'affluenza delle persone per ogni ora è visualizzata su un grafico lineare in cui l'utente può osservare i momenti della giornata con maggiore afflusso.
I dati vengono prelevati dal cloud usando sempre come mezzo di comunicazione le API di Measurify.

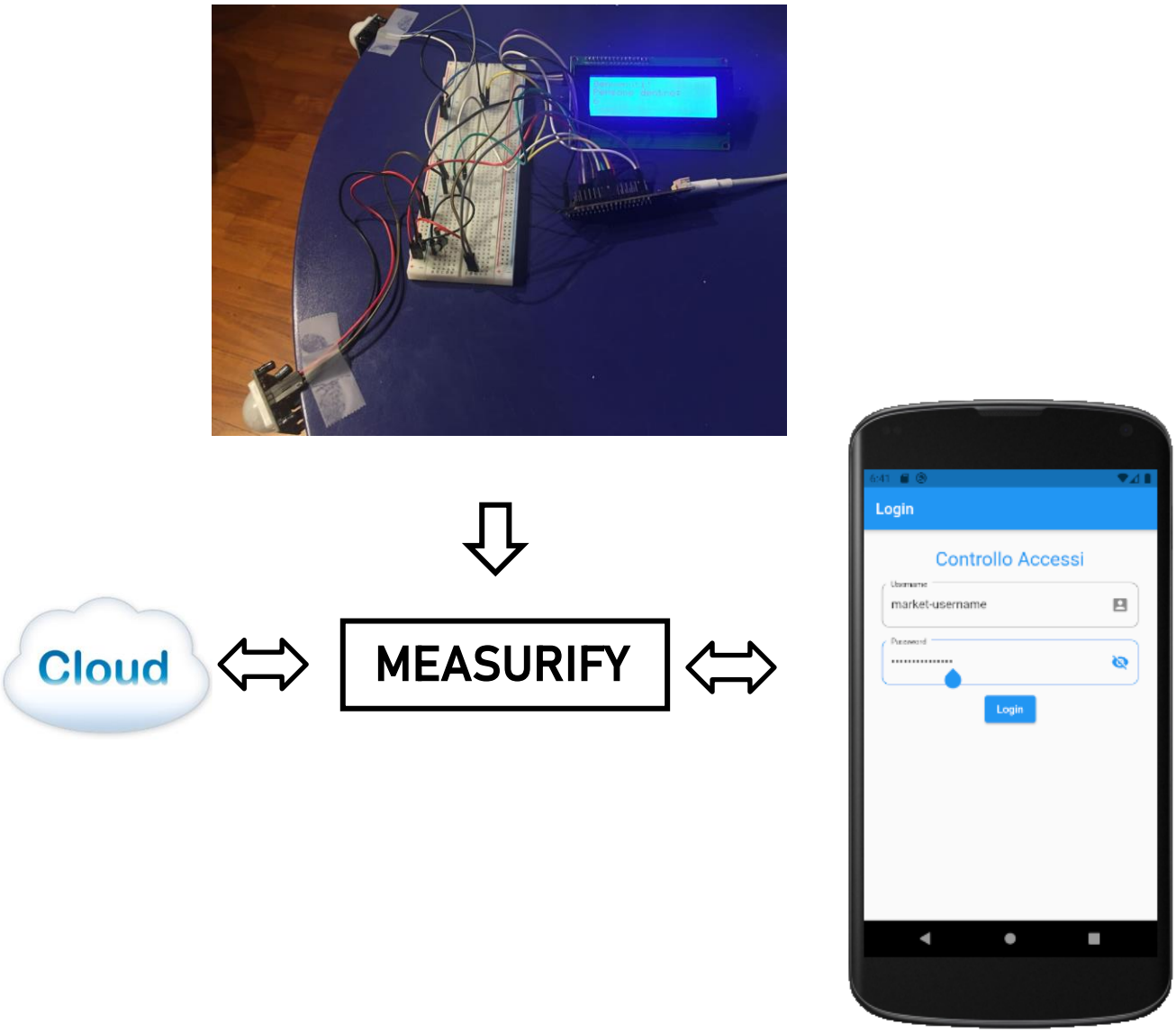

Figura 2: Schema esemplificativo del progetto

2. Metodi e strumenti utilizzati

2.1 ESP32-WROVER-B

Alla base del progetto hardware è stato utilizzato il microcontrollore ESP32, prodotto dalla società Espressif Systems. Quest'ultimo è il successore del chip Wi-Fi ESP8266 che nel 2014 aveva avuto molto successo perché permetteva, a basso costo, di connettere microcontrollori, come, ad esempio, Arduino alle reti Wi-Fi. Come il suo predecessore, anche l'ESP32 ha il supporto integrato per la connessione al Wi-Fi che è stata sfruttata per connettersi con le API di Measurify. Le caratteristiche di questo microcontrollore sono le seguenti:

- Processore Tensilica LX6 dual core a 240 MHz con Architettura Harvard
- 520Kb di memoria SRAM
- Connettività Wi-Fi 802.11 b/g/n (supporto per WEP, WPA/WPA2 PSK/Enterprise)
- Connettività Bluetooth (classica e LE)
- 32 PIN di I/O con diverse periferiche disponibili
- Accelerazione hardware per algoritmi di sicurezza (AES, SHA2, RSA-4096)
- Periferiche: SPI, I2C, I2S, UART, CAN 2.0 e interfaccia Ethernet;

La scheda utilizzata è la ESP32-wrover-B che è dotata di un regolatore di tensione per abbassare i 5V forniti via USB ai 3.3V necessari per il funzionamento del chip.

Come ambiente di sviluppo è stato utilizzato Arduino IDE che, come si può evincere dal nome, è l'ambiente di sviluppo di Arduino. Per poterlo usare con questa scheda è stato necessario scaricare il pacchetto specifico per la programmazione con i moduli ESP32-wrover.

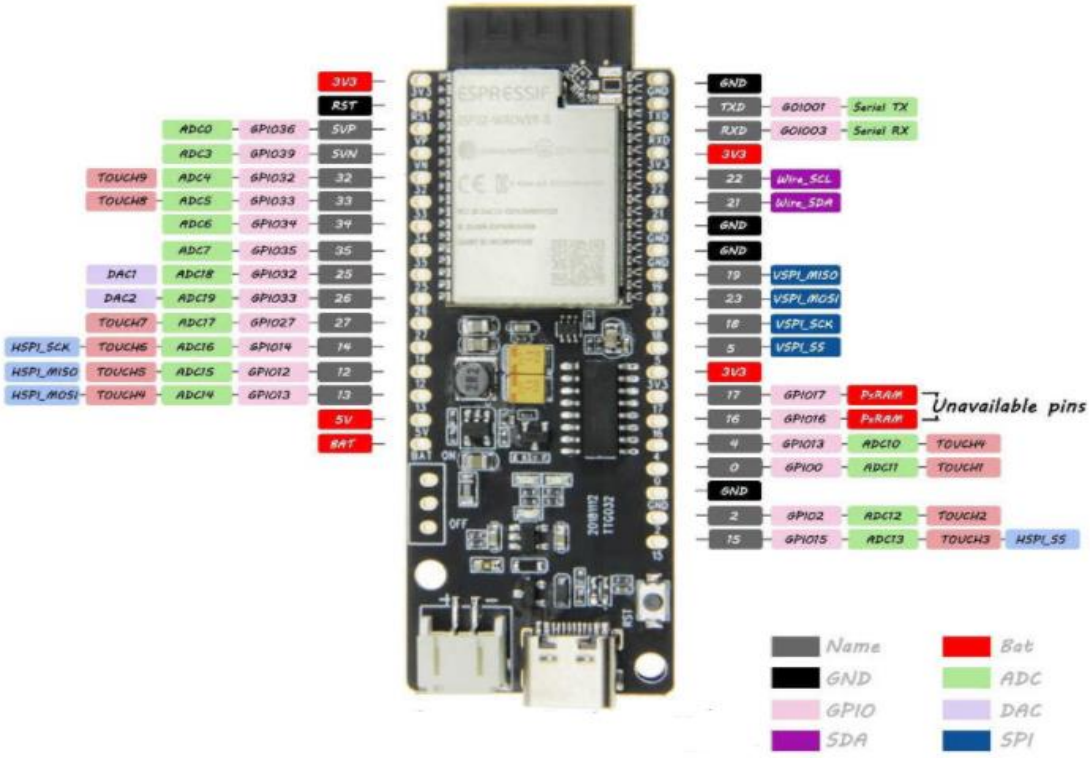

Figura 3: Pinout dell'ESP32-WROVER-B

2.1. Sensore PIR HC-SR501

La sigla PIR sta per Passive Infra-Red, ovvero infra-rosso passivo ed è un tipo di sensore che sfrutta un dispositivo piroelettrico che rileva il movimento misurando il cambiamento dei livelli di infrarossi emessi dagli oggetti che circondano il sensore. Il movimento viene rilevato da un sensore infrarosso passivo di tipo LHI778 che invia il segnale ad un circuito integrato di tipo BISS0001 che, a sua volta, rileva se vi è movimento o no. Il seguente modulo invia un'informazione di tipo bit in uscita:
- 0 per l'assenza di movimento
- 1 se c'è stato movimento

La parte anteriore è formata da una semisfera di plastica al cui centro vi è il sensore infrarosso; questa semisfera raccoglie verso il sensore tutte le fonti infrarosse.

La parte posteriore del sensore è la parte contenente il circuito formato da:

- Regolatore di tensione a 3.3V per l'alimentazione.
- Circuito integrato BISS0001 citato in precedenza

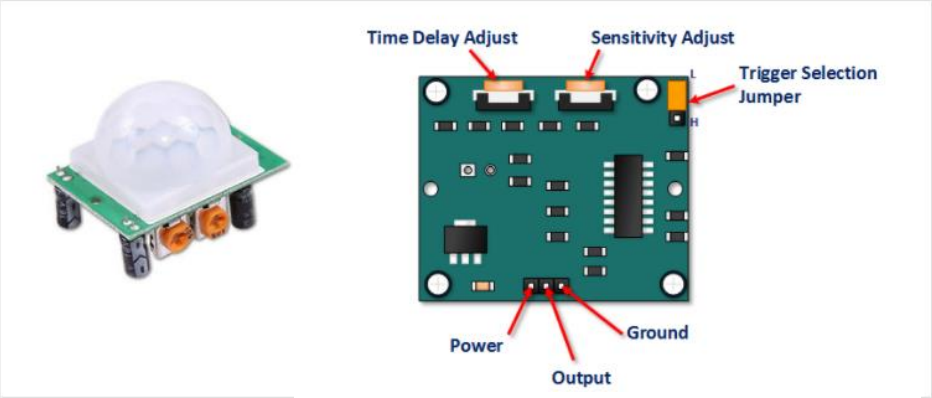

Figura 4: Struttura del sensore PIR

- Pin di alimentazione, output e GND.
- Controllo del delay, ovvero di quanto rimane alto il segnale dopo il movimento, regolabile attraverso il primo trimmer da un minimo di 3 secondi ad un massimo di 5 minuti. Il delay si aumenta girando in senso orario.
- Controllo della sensibilità, regolabile attraverso il secondo trimmer da un minimo di 3 metri ad un massimo di 7
- Jumper con due posizioni. Se il pin centrale è connesso al **pin L**: dopo il movimento il segnale diventa alto e rimane alto per un tempo che dipende dal trimmer del tempo; se vi è un nuovo movimento mentre il segnale è alto il tempo non viene resettato. Se il pin centrale è connesso al **pin H**: il pin di uscita del segnale rimane a livello alto dopo l'ultimo movimento per un tempo che dipende dal trimmer del tempo, indipendentemente se prima vi è stato un movimento o no.

Caratteristiche generali:

- Tensione alimentazione: 5V fino a 20V massimi
- Consumo di corrente: 65mA – 50uA in sleep
- Distanza massima rilevamento: 7m
- Livelli tensione uscita: 0V no movimento, 3.3V movimento, TTL
- Durata segnale: da 3 secondi a 5 minuti regolabili
- Temperatura funzionamento: -15°C/70°C
- Angolo di sensing 110°

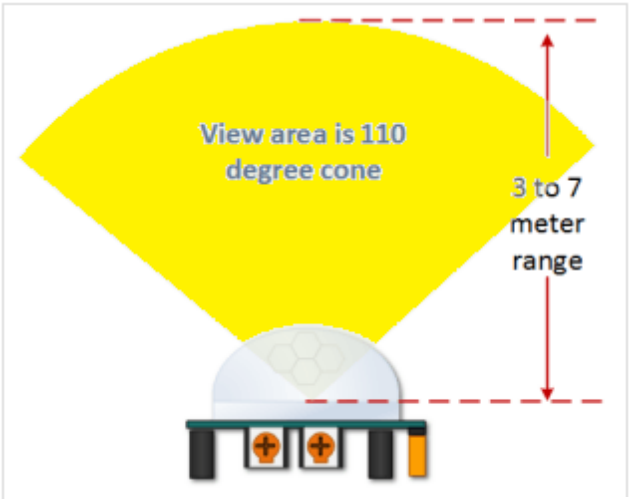

Figura 5: Caratteristiche generali del sensore PIR

2.2. Display LCD

Per la visualizzazione dei dati in loco è stato utilizzato un **display LCD 20x04**, con alimentazione a 5V, dotato di un adattatore che ridurrà il numero di connessioni necessarie e fornirà la possibilità di sfruttare l'interfaccia I²C. Quest'ultimo è un convertitore dal bus seriale I²C al bus parallelo utilizzato dai Display LCD con standard HD44780 e matrice (colonne x righe) nel nostro caso 20x04. Lo schema elettrico è basato sul chip PCF8574 un I/O expander per I²C che si occupa di convertire i dati provenienti dalla linea dati e pilotare le linee del display. Sul modulo sono presenti:

- Un trimmer per la regolazione del contrasto
- Un jumper rimovibile per l'attivazione della retroilluminazione
- 4 pin di collegamento per I²C
- I tre pin A0 A1 A2 per utilizzare più di un display
- E i 16 pin per il collegamento all'LCD

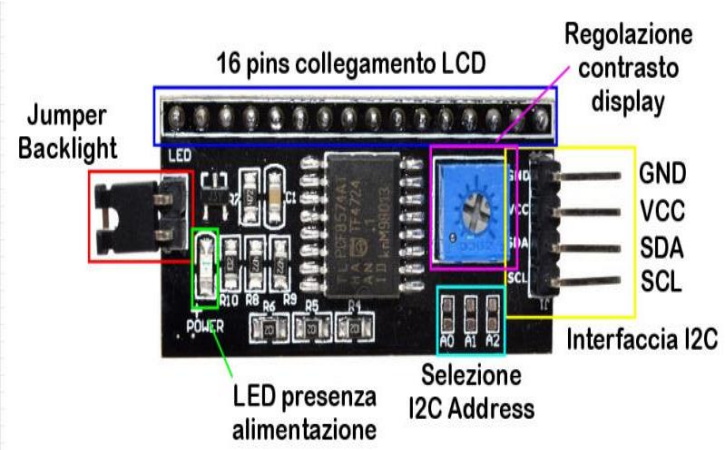

Figura 6: Adattatore

E' nessario collegare:
-il pin GND al GND della scheda
-il Vcc al pin 5v di alimentazione
-il pin SDA al pin 21
- l'SCL al pin 22

I pin 21 e 22 sono i due pin della scheda predisposti per il protocollo I²C come si può vedere dal pinout mostrato in precedenza. Per poter utilizzare LCD sono state aggiunte al progetto la libreria Wire.h per gestire l'interfaccia I²C e la libreria LiquidCrystal_i2C.h per gestire il display. In seguito è stato creato l'oggetto lcd specificando l'indirizzo(0x27) e le dimensioni(20, 4).

```
1  #include <Wire.h>
2  #include <LiquidCrystal_I2C.h>
3
4  LiquidCrystal_I2C lcd(0x27, 20, 4);
```

Figura 7: Librerie

Protocollo I²C

Il protocollo I²C è stato creato dalla Philips Semiconductors nel 1982. L'acronimo sta per: Inter-Integrated Circuit. L' I²C è un sistema di comunicazione seriale bifilare utilizzato tra circuiti integrati. Per abilitare la connessione sono necessari 4 pin:

- Alimentazione VCC (5V)
- Tensione di massa GND
- SDA(**S**erial **DA**ta) per i dati
- SCL(**S**erial **Cl**ock) per il clock, che rende l' I²C un bus sincrono.

Il bus ha due tipi di nodi, il master e lo slave. Il primo emette il segnale di clock mentre il secondo si sincronizza sul segnale di clock senza poterlo controllare. Nel bus possono essere presenti più

dispositivi che possono svolgere la funzione di master ma solo uno alla volta. In generale ci sono 4 diversi modi di operare:

- un master trasmette, controlla il clock e invia dati agli slave
- un master riceve, controlla il clock ma riceve dati dallo slave
- lo slave trasmette, il dispositivo non controlla il clock ma invia dati al master
- lo slave riceve, il dispositivo non controlla il clock ma riceve dati dal master.

Il **master** ha il compito di iniziare o terminare lo scambio di informazioni inviando rispettivamente il segnale di START (S) seguito dall'indirizzo dello slave con cui vuole comunicare (L' I^2C ha 7 bit di indirizzamento dal più significativo B1 al meno significativo B7) o il segnale di STOP(P). I due segnali hanno particolari caratteristiche per essere riconosciuti:

- lo Start (S) è costituito da una transizione da alto a basso del bus dati SDA mentre il clock (SCL) è alto
- lo Stop (P) è rappresentato da una transizione da basso ad alto dell'SDA mentre l'SCL è alto.

Nel caso in cui il segnale sia di START dopo i 7 bit di indirizzamento, il bit B8 indica se il master vuole trasferire informazioni allo slave (scrivere, ammesso che il dispositivo permetta questa possibilità) o ricevere informazioni da esso (leggere). Nel primo caso il bit B8 è tenuto basso dal master mentre nel caso in cui voglia ricevere informazioni il bit B8 sarà alto.

2.3. Measurify

Measurify è un framework open source per lo sviluppo di applicazioni IoT che si occupano di misurazioni, in questo progetto derivanti da sensori, sviluppato nell'Università di Genova da Elios Lab.
Measurify è stato progettato per rappresentare il contesto dell'applicazione e i suoi elementi come oggetti software correlati, sui quali costruire applicazioni.
Questo Framework è strutturato in modo tale da essere velocemente implementabile partendo dal modello dei dati dell'applicazione che si ha intenzione di programmare basandosi su alcuni concetti di base:

- **THING**: si tratta di una "cosa" (una persona, un ambiente, un oggetto, etc.) per la quale si sta facendo una misurazione. In questo progetto è il supermercato.

- **FEATURE**: si tratti di che cosa si sta misurando (la temperatura, l'umidità, il peso, etc.). In questo progetto sono stati misurati gli ingressi e le uscite dal supermercato.

- **DEVICE**: si tratta di un dispositivo (hardware/software) che può misurare una certa dimensione (feature) su una determinata cosa (thing). In questo progetto il device è rappresentato dall'insieme di board con microcontrollore e sensori PIR.

- **MEASUREMENT**: si tratta di una misura effettuata da un dispositivo (device) per una certa dimensione (feature) su una determinata cosa (thing). La measurement contiene tutte le informazioni sulla misura che è stata effettuata. Indica l'inizio e la fine della misura, la thing a cui si riferisce la misura, il device che l'ha rilevata, la feature che rappresenta e il valore della misurazione indicato in un array di samples.

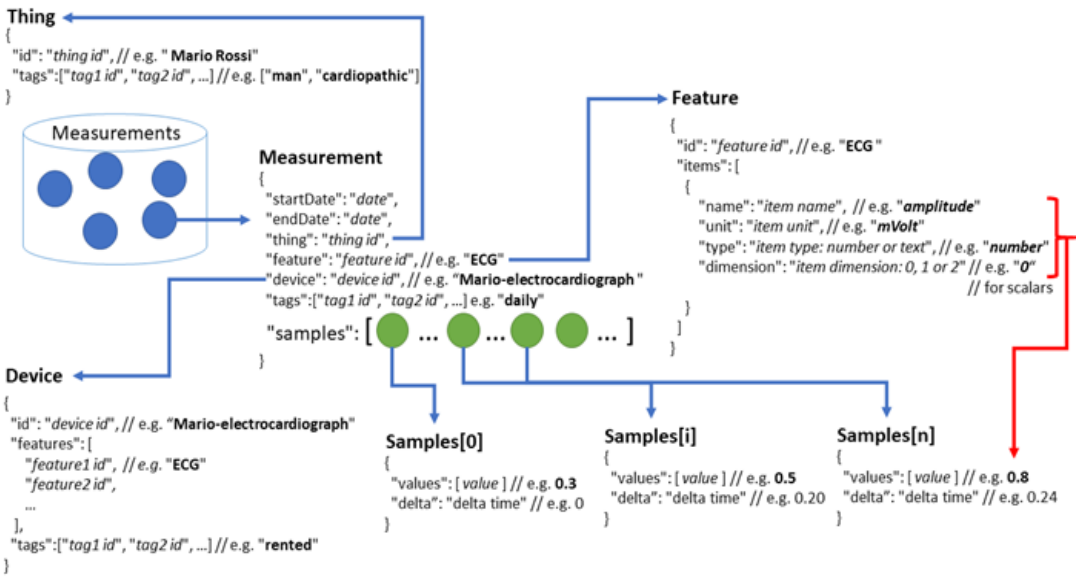

Figura 8: Modellizzazione delle risorse

I dati così organizzati saranno archiviati in un database che l'utente potrà interrogare utilizzando i verbi supportati da HTTP (GET, POST, PUT, DELETE). Tra l'interfaccia utente e il database è posta un'API di tipo RESTful.

Durante lo sviluppo di questo progetto è stata utilizzata l'**applicazione Postman** per interrogare il database e testare sia il sistema embedded che l'applicazione smartphone. Per effettuare le richieste al database è necessaria l'autenticazione. Quindi utilizzando Postman è stata creata una POST alla url:

https:\\students.atmosphere.tools/v1/login

Con il body decorato con i dati della login:

```
{
"username" : "market-username",
"password" : "market-password"
}
```

Figura 9: Dati della login

Se la POST va a buon fine viene restituito un JSON con all'interno un token, valido per 30 minuti, il quale andrà inserito in tutte le altre richieste HTTP come header Authorization. Per fare questo è stata inserita nella parte di Tests della POST/login una riga di codice che estrae il token dalla risposta HTTP e lo salva in una variabile che sarà inserita nell' header Authorization.

Eseguito il login sempre attraverso Postman sono state usate due POST per inserire nel database measurements di un ingresso o di un'uscita alla url:

https:\\students.atmosphere.tools/v1/measurement

```
postman.setEnvironmentVariable("token",JSON.parse(responseBody).token);
```

Con il seguente body:

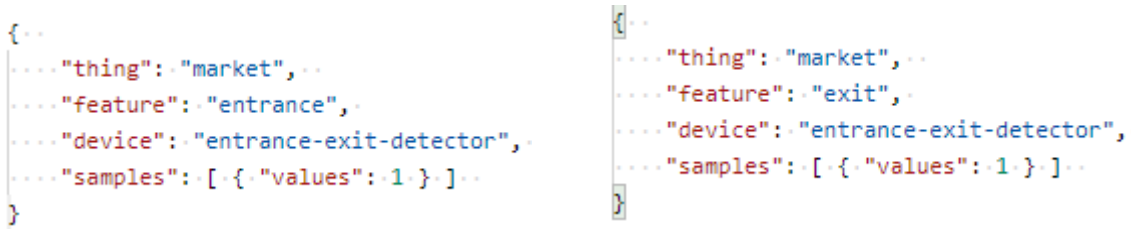

Figura 10: Dati ingresso-uscita

Da questi body si nota che per inserire il dato facciamo sempre riferimento alla modellizzazione dei dati fatta con Measurify.

In modo analogo, è stata fatta una GET sulla stessa url delle due POST precedenti per ottenere l'elenco delle measurements. Applicando dei filtri in cui si specifica il valore del campo desiderato, questa GET risulta fondamentale nella realizzazione dell'applicazione perché è quella che viene richiamata ogni volta che si vuole calcolare il numero di persone presenti all'interno del negozio considerato.

2.4. Edge Engine

Si tratta di una applicazione scritta in codice C standard che può essere compilata per board con microcontrollore e per piattaforme desktop. L'idea è di avere una libreria che aiuti a interagire con le API di Measurify per supportare il programmatore ad eseguire della computazione direttamente su microcontrollore, descrivendo però la computazione sul cloud. Quest'applicazione può elaborare flussi di dati dai sensori tramite script o eseguire comandi utilizzando gli attuatori ad esso collegati. Gli script sono costituiti da un insieme di operazioni note che possono essere utilizzate anche insieme per eseguire calcoli complessi su ogni flusso di misura. Edge Engine può essere configurato per recuperare gli script online (da un cloud) da eseguire localmente e controllare periodicamente se questi sono cambiati o se ci sono nuovi script da eseguire. Non è legato ad alcuna piattaforma proprietaria, il provider può scegliere il servizio cloud a cui sarà connesso l'engine. In questo progetto è stato connesso alle API di Measurify.
Per utilizzare Edge Engine nel progetto è stato compilato sul dispositivo fisico che raccoglie i dati che in questo caso è la board con microcontrollore ESP32. Dopo di che è stata inserita la libreria apposita nello sketch attraverso il comando "#include <EdgeEngine_library.h>".
Per sfruttare questa libreria serve inoltre che ogni dispositivo fisico del progetto sia associato ad un identificativo univoco (che deve essere inserito all'interno del codice dell'Edge Engine) per poter accedere alla sua rappresentazione virtuale. Sempre utilizzando Postman, questa rappresentazione la si ottiene con una GET alla url:

https:\\students.atmosphere.tools/v1/devices/entrance-exit-detector

Si ottiene come risposta il JSON riportato qui a lato. Da questo si possono dedurre tutte le caratteristiche del dispositivo. Tra le più importanti troviamo le **"features"** che sono le grandezze che il dispositivo è capace di misurare (in questo progetto sono gli ingressi e le uscite) ma anche altre informazioni, come il nome degli scripts, in modo tale da ottenerne il codice con una GET apposita.

Per poter comunicare con Measurify attraverso Edge Engine è necessario anche in questo caso effettuare il login e quindi anche questi dati andranno inseriti nel codice dell'engine.


```
{
    "features": [
        "entrance",
        "exit"
    ],
    "tags": [],
    "scripts": [
        "simple-entrance",
        "simple-exit"
    ],
    "visibility": "private",
    "period": "5s",
    "cycle": "10m",
    "retryTime": "10s",
    "scriptListMaxSize": 5,
    "measurementBufferSize": 20,
    "issueBufferSize": 20,
    "sendBufferSize": 20,
    "scriptStatementMaxSize": 5,
    "statementBufferSize": 10,
    "measurementBufferPolicy": "newest",
    "_id": "entrance-exit-detector"
}
```


Figura 11: Rappresentazione virtuale entrance-exit detector

2.5. Flutter

Flutter è un framework open-source creato da Google con lo scopo di sviluppare applicazioni native per IOS e Android e anche applicazione Web. Come editor è stato utilizzato Visual Studio Code, realizzato da Microsoft, il quale implementa e si interfaccia con l'SDK. Come emulatore è stato utilizzato l'emulatore fornito da Android Studio.
Flutter si basa su questi elementi principali:

- **Dart platform:** E' un linguaggio di programmazione sviluppato da Google con lo scopo di sostituire JavaScript per lo sviluppo Web. Il compilatore di Dart permette di scrivere programmi sia per web che per desktop, smartphone e server, tramite due diverse piattaforme:
    - Dart Native pensata per smartphone, server, desktop e altro
    - Dart Web pensata appositamente per il web

- **Flutter Engine:** Scritto principalmente in C++, fornisce ciò che rende possibile il rendering di basso livello utilizzando la libreria grafica di Google, Skia Grapichs. Una delle funzionalità più apprezzate di questo motore, grazie al codice scritto in Dart, è nominata "hot-reload" e permette, in fase di sviluppo, di mostrare le modifiche apportate senza il bisogno di riavviare l'applicazione.

- **Foundation Library:** Si tratta di una libreria scritta in Dart, la quale fornisce tutte le classi e i metodi di base per sviluppare applicazioni tramite Flutter.

- **Widgets:** Il codice sviluppato con Flutter si basa sui widgets, i quali sono descrizioni immutabili dell'interfaccia utente, alcuni esempi possono essere testi, bottoni, forme, grafici e animazioni. I widgets sono una delle cose che rendono lo sviluppo con Flutter semplice e intuitivo, poiché, se si è interessati a creare parti più complesse, basta combinare widgets elementari di diverso tipo.

3. Sperimentazione e Risultati

3.1.Sistema embedded

3.1.1. Cablaggio

Alla base del progetto di questa tesi c'è la scheda di sviluppo ESP32-wroover-B che ha il compito insieme ai sensori ad essa collegati di raccogliere i dati necessari e inviarli al cloud. Attraverso una breadbord e dei cavetti jumper sono stati eseguiti i collegamenti necessari tra board con microcontrollore e sensori. Di seguito saranno indicati sulla sinistra i pin dei sensori mentre sulla destra i relativi pin della board:

- Sensore PIR ingresso
  GND → GND
  OUTPUT → PIN 35
  VCC → 3V

  Sensore PIR uscita
  GND → GND
  OUTPUT → PIN 34
  VCC → 3V

- Chip PCF8574
  GND → GND
  VCC → 5V
  SDA → PIN 21
  SCL → PIN 22

Da notare che i pin collegati ai sensori PIR sono pin che possono gestire gli interrupts mentre i pin collegati ai pin SDA e SCL del chip sono quelli predisposti dalla scheda ESP32 -wroover-B per gestire l'interfaccia I^2C.

3.1.2. Fase di setup

Per programmare la board con microcontrollore è stato scritto uno sketch sull'IDE predisposto per Arduino utilizzabile anche con schede dotate di ESP32 come quella utilizzata in questo progetto. Lo sketch è diviso in due fasi caratterizzate rispettivamente dalla funzione di setup e da quella di loop.

Prima della fase di setup vengono definite le variabili che saranno utilizzate nelle successive funzioni dello sketch. Le variabili definite saranno utilizzabili in qualunque parte del suddetto sketch. In questo punto sono anche associate le variabili ai pin che rappresenteranno, in modo tale che ogni volta che ci si vorrà riferire ad uno specifico pin, si potrà utilizzare la variabile predisposta per maggior chiarezza.

La **fase di setup** è caratterizzata dalla funzione omonima. Questa particolare funzione viene eseguita solo una volta appena lo sketch è caricato sul microcontrollore. Come prima cosa è stato inizializzato l'LCD e sono state riempite le righe (del display dell'LCD) che saranno sempre fisse durante tutta l'esecuzione. Successivamente viene inizializzata la connessione al Wifi. A questo punto viene decorata la struttura "opts" che contiene tutte le variabili che descrivono il modello del dispositivo fisico e che saranno assegnate alla sua rappresentazione virtuale. Infine, dopo aver inizializzato l'istanza di Edge Engine passando la struttura appena descritta, vengono definite le modalità con cui funzioneranno i pin. Attraverso la funzione pinMode i due pin dei sensori PIR vengono riconfigurati come ingressi. Successivamente questi due pin vengono passati al comando *attachInterrupt*.

```
attachInterrupt(digitalPinToInterrupt(pirPin_entrance), detectedEntrance, RISING);
attachInterrupt(digitalPinToInterrupt(pirPin_eXit), detectedExit, RISING);
```

Questa istruzione specifica che nel momento in cui uno dei sensori PIR rileverà un movimento, il programma gestirà questo evento in modo asincrono attraverso la tecnica dell'interrupt. Quest'ultima prevede che una periferica, in questo caso il sensore PIR, possa inviare un segnale di richiesta di interruzione che, quando viene ricevuto dal microcontrollore esso, finita l'istruzione che sta eseguendo, passa il controllo alla Interrupt Service Routine (ISR). L'istruzione *attachInterrupt()* prende in ingresso tre parametri:
1. Come primo parametro bisogna passare il numero di interrupt a cui è collegato il sensore. Per facilitare questo passaggio viene usato il comando *digitalPinToInterrupt()* che traduce il pin digitale che gli viene passato nel numero di interrupt specifico.
2. Il secondo parametro è la funzione che viene chiamata quando si verifica l'interrupt.
3. L'ultimo parametro è quello che definisce il valore che deve avere il segnale per far partire l'interruzione. Questo parametro può avere 4 valori:

- **LOW:** l'interrupt viene eseguito quando il livello del segnale è basso
- **CHANGE**: l'interrupt viene eseguito quando avviene un cambiamento di stato sul pin
- **RISING:** l'interrupt viene eseguito quando si passa da un livello LOW ad un livello HIGH
- **FALLING:** l'interrupt viene eseguito quando si passa da un livello HIGH ad un livello LOW

3.1.3. Fase di loop

La seconda fase dello sketch è costituita dalla **funzione loop**. Questa funzione, come si può dedurre dal nome, verrà ripetuta in loop per tutta la durata d'esecuzione dello sketch. Di seguito è stato inserito un diagramma di flusso che descrive gli step di questa funzione:

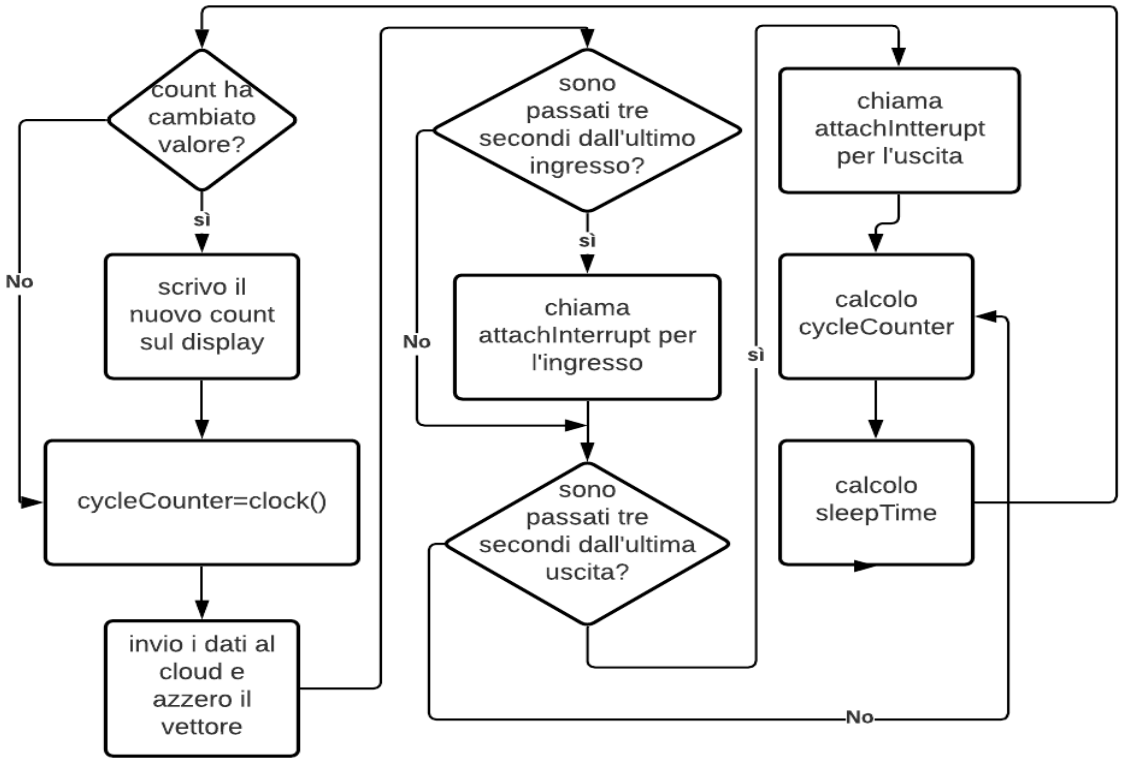

Figura 12: Schema di flusso della funzione di loop

- **Controllo la variabile count:** Il primo comando della fase di loop è un *if().* Questo serve per fare in modo che Il display dell'LCD venga aggiornato solo nel caso in cui la variabile count, che è quella predisposta per tenere conto delle persone presenti all'interno del negozio d'esempio, sia cambiata.

- **Aggiorno il display:** Se la variabile count è cambiata questo implica che c'è stato un ingresso/uscita e quindi bisognerà aggiornare il numero rappresentato sul display.

- **Assegno il valore alla variabile cycleCounter:** A questo punto eguagliamo la variabile cycleCounter alla funzione clock(). Questa funzione restituisce il numero di "tick" della CPU utilizzati dal processo sin da quando è partito. Questo servirà in seguito per calcolare lo sleep time alla fine della funzione loop.

- **Invio i dati al cloud:** Successivamente attraverso il comando evaluate della libreria < EdgeEngine_library.h > vengono inviati al cloud i dati raccolti nel ciclo del loop precedente a quello corrente. Poi con il comando clear viene azzerata la variabile che contiene i dati per liberarla per il ciclo di loop successivo.

- **Controllo se il sensore è pronto per rilevare un movimento:** Essendo che il delay time dei due sensori PIR è impostato al minimo (3 secondi) e il trigger selection jumper è settato su L, dopo il rilevamento di un movimento il segnale del sensore rimane alto per 3 secondi. Durante questi 3 secondi qualsiasi movimento non può essere rilevato. Quindi sempre sfruttando la funzione *clock()* come parametro si verifica che siano passati 3 secondi dall'ultima misurazione. Se così non fosse, in questo ciclo del loop il sensore viene "silenziato".

- **Chiamo la funzione attachIntterupt:** Invece se questo tempo è passato, viene chiamata la funzione *attachInterrupt* che prenderà il controllo nel momento in cui verrà rilevato un movimento e lo passerà alla Interrupt Service Routine (ISR). Questo comando, come già detto, è asincrono quindi lo sketch non si fermerà a questo ma procederà col successivo. Sarà L'interrupt che fermerà il processo nel momento in cui verrà rilevato un movimento. Durante l'Interrupt Service Routine viene eseguita la funzione:
  - *detectedEntrance()* nel caso sia stato rilevato un ingresso
  - *detectedExit()* nel caso in cui sia stata rilevata un'uscita
  In queste due funzioni come prima cosa viene aggiornata la variabile count. Successivamente viene creata la variabile che conterrà i dati della misura rilevata. Subito dopo questa variabile viene decorata con tutte le informazioni da assegnare alla misura. Infine essa viene inserita in un vettore che sarà quello che verrà usato per inviare i dati al cloud durante il successivo ciclo del loop. Queste ultime due parti sono ripetute una volta per il sensore che si occupa degli ingressi e una volta per quello delle uscite.

- **Assegno nuovo valore alla variabile cycleCounter:** Quando il controllo torna alla funzione di loop esso attraverso l'operazione *cycleCounter = clock()-cycleCounter* calcola il tempo che è stato impiegato dai processi compresi tra questo comando e la prima assegnazione del *clock()* a cycleCounter (avvenuta nel terzo blocco di questo schema).

- **Calcolo sleepTime:** Come ultimo comando di ogni ciclo viene calcolato il tempo di attesa tra un ciclo e l'altro, in base alla variabile cycleCounter e al delay time dei sensori PIR e viene lanciato il comando *delay()* che mette in pausa il processo per il tempo indicato in millisecondi dalla variabile che gli viene passata.

3.2. Applicazione smartphone

L'applicazione smartphone è sicuramente di fondamentale importanza per questo progetto. Dopo aver raccolto i dati in loco ed averli salvati sul cloud, dare la possibilità agli utenti dell'applicazione di consultare questi dati è ciò che rende significativo e tangibile il progetto. L'applicazione è stata programmata grazie all'utilizzo di un framework open source e cross-platform chiamato Flutter. Come linguaggio di programmazione è stato utilizzato **Dart** mentre come IDE Visual Studio Code. Come emulatore è stato utilizzato quello fornito da Android Studio e il telefono che è rappresentato nelle immagini è un Nexus 4. L'applicazione è suddivisa in 3 schermate:

- **Login**
- **I miei negozi**
- **Negozio**

3.2.1. Login

Quando l'applicazione viene aperta la prima pagina visualizzata è la pagina di login. Nella appBar è stato inserito il titolo della pagina che è appunto login. Subito sotto in un widget Text è inserito il nome dell'applicazione. Sotto sono presenti due TextFormFild che sono i campi predisposti per inserire username e password. In questi due widgets sono inserite due icone ma, mentre la prima è una semplice icona, la seconda rappresenta anche un bottone. Alla pressione dell'icona nel campo password si può passare dalla modalità con testo leggibile alla modalità con il testo oscurato.

Come ultimo widget della colonna è presente un bottone che, se premuto, innescherà alcuni controlli prima di tentare la POST login. Come prima cosa dopo la pressione del tasto login vengono chiamati i campi *validator* dei due widgets TextFormFild. In questo campo sono stati inseriti il controllo per verificare che il campo non sia stato lasciato vuoto e quello che verifica che il testo inserito non sia inferiore ai tre caratteri (presupponendo che sia lo username che la password debbano avere almeno quattro caratteri).

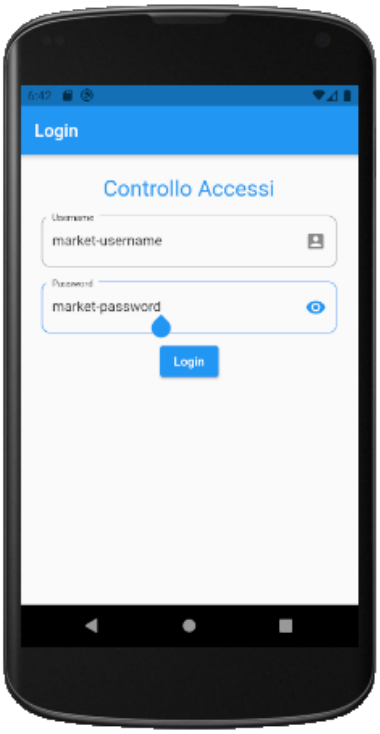

Figura 13: Pagina di login

Se questi controlli non vengo superati appariranno nel campo dove è stato riscontrato un errore il relativo messaggio. Viceversa, se le informazioni richieste sono state inserite correttamente si procede al salvataggio del testo dei due campi in due variabili apposite. L'ultimo passaggio consiste nella chiamata alla funzione login. Prima di parlare di questa funzione bisogna precisare che i dati richiesti per accedere all'app sono gli stessi utilizzati per accedere alle API di Measurify. A questo punto chiamata la funzione login con username e password come parametri, essa provvede a comporre la chiamata POST con, all'interno del body, in formato JSON, i due dati inseriti dall'utente. Come prima cosa dopo la ricezione della risposta viene fatto un controllo sul suo statusCode. Se questo è diverso da 200 viene aperto un popup che avvisa l'utente che o username o password sono errati. Nel caso in cui lo statusCode sia uguale a 200 la funzione di login provvede a salvare il campo token in una variabile e far accedere il cliente alla seconda pagina dell'applicazione.

3.2.2. I miei Negozi

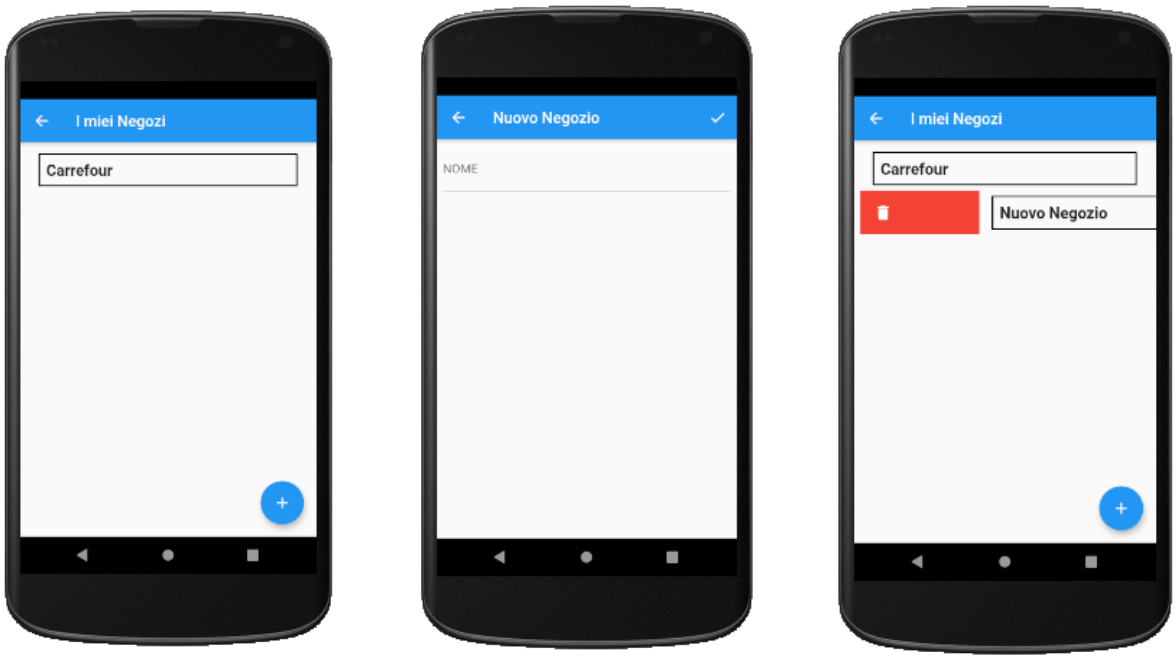

Figura 14: Pagina con la lista dei negozi

La seconda pagina dell'applicazione è quella predisposta per contenere la lista dei luoghi, in questo esempio negozi, che l'utente può consultare. Facendo un click sul bottone in basso a destra si apre la pagina che ci permette di inserire un nuovo negozio. Inserito il nome nel campo basterà cliccare sulla spunta in alto a destra per confermare l'inserimento. Cliccando invece sulla freccia in alto a sinistra si torna alla pagina precedente annullando l'inserimento. Se l'utente ha inserito il nome e cliccato sulla spunta verrà riportato alla pagina contenente la lista dei negozi e lì troverà il nuovo inserimento. Facendo click sul nuovo negozio e scorrendo verso destra potrà cancellare il negozio inserito. Facendo click su uno dei negozi invece accede alla terza pagina dell'applicazione.

3.2.3. Negozio

La terza e ultima pagina dell'applicazione è quella che permette di consultare i dati rilevati dal sistema embedded. La schermata si divide in tre parti:

1. La parte superiore è quella che riguarda il momento in cui l'utente sta interagendo con l'applicazione. Appena si apre la pagina, il campo di testo è inizializzato con 3 tratti orizzontali in modo da far notare il punto in cui apparirà il numero di interesse. Dopo aver premuto il tasto, il sistema prende la variabile in cui sono salvati la data e l'ora del momento della pressione del tasto e li passa ad una funzione che chiamerà le richieste GET HTTP. Le richieste saranno due: una per gli ingressi e una per le uscite. La url di queste richieste sarà caratterizzata da alcuni filtri che specificano cosa si vuole riceve dal database. Prima di tutto sarà specificato il **campo feature** che indicherà il tipo di rilevazione a cui si è interessati (ingresso/uscita). In secondo luogo, viene specificato l'**intervallo di tempo** del quale si vuole ricevere le misurazioni. Ricevuta la risposta dal database, viene controllato che lo statusCode sia uguale a 200 e in caso affermativo viene estratto il body della risposta. La risposta che arriva sarà in formato stringa quindi verrà parsata, attraverso il comando jsonDecode, in un oggetto JSON. A questo punto da questo oggetto vengono ricavati il numero di rilevazioni sia per gli ingressi che le uscite. Ottenuti questi dati, basterà farne la differenza e mostrare a schermo il risultato ottenuto. Essendo richieste HTTP verso le API di Measurify ogni

richiesta dovrà essere decorata con l'header di autorizzazione con al suo interno il token ottenuto dalla POST/login. Tutte queste funzioni contengono operazioni bloccanti quindi di base sarebbero funzioni asincrone. In questo caso specifico però vengono rese sincrone attraverso l'utilizzo del comando *async/await* perché senza i dati estratti dal database non si avrebbero le informazioni per proseguire con l'esecuzione del programma.

2. La seconda parte della schermata invece consente all'utente di conoscere il numero di persone presenti all'interno in un momento stabilito dall'utente stesso. Facendo click sulle icone si apriranno un calendario ed un orologio che permetteranno all'utente di scegliere data e ora di suo interesse. Le operazioni innescate dalla pressione del tasto saranno simili a quelle descritte sopra. L'unica differenza è che, in questo caso, i dati su data e ora andranno estrapolati dopo la decisione dell'utente.

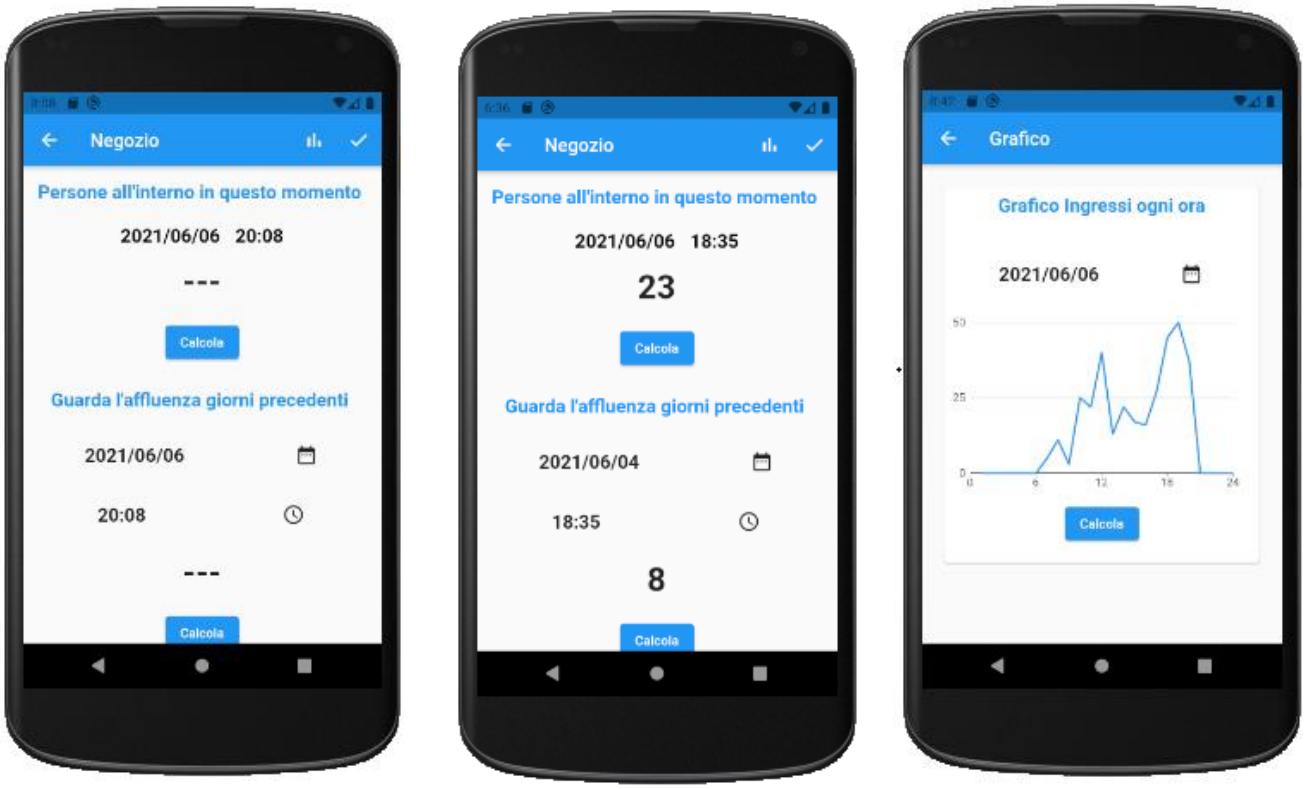

Figura 15: Pagina del negozio selezionato

3. Premendo l'icona in alto a destra raffigurante un diagramma, si apre una terza parte della schermata in cui è presente un grafico lineare. Quest'ultimo rappresenta il numero di ingressi per ogni ora del giorno selezionato dall'utente tramite l'apposita icona. Sull'asse delle ascisse sono inserite le ore della giornata (0-23h) e su quello delle ordinate il numero di persone. Questa funzionalità risulta utile all'utente per visualizzare in modo più rapido e diretto gli orari con maggiore affluenza di clienti.

4. Contributo personale e considerazioni conclusive

Questa tesi ha l'obiettivo di creare un progetto che si inserisca nell'ambito sempre più in espansione dell'IOT. Lo scopo è stato quello di implementare un sistema integrato che avesse la capacità di connettersi alla rete per trasmettere i dati da esso rilevati. Questo è stato possibile attraverso l'utilizzo di un microcontrollore ESP32 montato su una scheda con modulo WiFi integrato. Raccolti i dati essi sono stati inviati e salvati sul cloud attraverso l'utilizzo delle API di Measurify. Successivamente è stata programmata un'applicazione smartphone che permettesse di consultare i dati raccolti dal sensore andandoli ad estrarre dal cloud sempre attraverso l'utilizzo di Measurify.

Dalla lettura di questo elaborato si evince l'importanza che Measurify ha avuto in questo progetto. E' stato utilizzato sia per la parte hardware che per la parte software. In tutti e due gli ambiti il suo utilizzo è stato abbastanza semplice e intuitivo.

Un'altra applicazione molto utilizzata durante questo progetto è stata Postman. Essa ha permesso di testare le componenti del progetto attraverso chiamate HTTP senza l'utilizzo del browser. Con Postman è possibile creare il proprio ambiente dove istanziare variabili che poi possono essere utilizzate nella scrittura delle richieste HTTP. E' anche possibile creare una collezione di richieste in modo tale da averle sempre disponibili. L'ho personalmente utilizzato durante la programmazione del sistema embedded per controllare che i dati rilevati dal sensore venissero caricati in maniera appropriata. Postman è stato utilizzato anche durante la programmazione dell'applicazione per verificare che i dati ottenuti fossero attendibili.

Personalmente, ho riscontrato qualche problema nell'utilizzo del microcontrollore ESP32, legato al fatto che la documentazione online non fosse sufficiente e precisa, rispetto ad esempio a quella presente per il più conosciuto e utilizzato Arduino.

I possibili miglioramenti futuri da apportare riguardano sicuramente la seconda parte relativa all'applicazione smartphone.
Un primo spunto potrebbe essere la realizzazione di una lista di negozi o luoghi pubblici che supportano questo tipo di servizio in modo tale che l'utente possa controllare se il luogo a cui è interessato permetta questa funzionalità e inserirlo nella sua lista.
Infine, si potrebbe pensare di far agire l'applicazione in background in modo tale che essa, con un intervallo di tempo fissato dall'utente, possa fare delle GET sui negozi presenti nella lista. In questo modo l'applicazione potrebbe avvisare il cliente tramite delle notifiche sul momento migliore con minore affluenza per recarsi nel luogo di interesse.

Nello svolgimento di questo elaborato ho trovato particolarmente interessanti le sue possibili applicazioni pratiche: l'app infatti consentirebbe ai clienti di monitorare il numero di persone presenti all'interno degli esercizi commerciali preferiti, evitando così inutili code e attese nella frenesia della quotidianità. I campi di applicazione ovviamente potrebbero essere ampliati ad esempio a cinema, spiagge e luoghi comuni di frequentazione, potendo così scegliere il momento più opportuno in cui recarvicisi. All'interno del contesto attuale di emergenza sanitaria risulta inoltre utile per evitare assembramenti e quindi di notevole importanza per eventuali restrizioni future.

5. Riferimenti bibliografici

- Interrupt,https://www.maffucci.it/2012/06/11/appunti-su-arduino-interrupts/
- Sensore PIR, https://www.ne555.it/sensore-movimento-pir-hc-sr501/
- Measurify, https://measurify.org/
- Flutter, https://flutter.dev/
- ESP32, http://esp32.net/
- Dart, https://dart.dev/
- Arduino, https://www.arduino.cc/
- IOT, Materia del corso di studi: “Approccio Makers alla progettazione elettronica”
- Arduino, https://www.progettiarduino.com/
- Esp32, https://www.espressif.com/